FILE COPY

(2)

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

Pub
nee
Hea
Ma
r per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data
urden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington
lerson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of

AD-A233 799

1. RT DATE

3. REPORT TYPE AND DATES COVERED
Final: 03 Feb 1991 to 01 Mar 1993

4. TITLE AND SUBTITLE
Ada Compiler Validation Summary Report: Aitech Defense Systems Inc., AI-ADA/88K Version 2.4, VAXstation 3100 cluster (Host) to Tadpole TP880V(88100 based VME board)(bare machine) (Target), 900930W1.11030

5. FUNDING NUMBERS

6. AUTHOR(S)
Wright-Patterson AFB, Dayton, OH
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ada Validation Facility, Language Control Facility ASD/SCEL
Bldg. 676, Rm 135
Wright-Patterson AFB
Dayton, OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER
AVF-VSR-390.0291

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Pentagon, Rm 3E114
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Aitech Defense Systems Inc., AI-ADA/88K Version 2.4, Wright-Patterson AFB, OH, VAXstation 3100 cluster VMS Version 5.3 (Host) to Tadpole TP880V (88100 based VME board)(bare machine)(Target), ACVC 1.11.

DTIC
ELECTE
MAR 17 1991

14. SUBJECT TERMS
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFIED	

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 30 September 1990.

Compiler Name and Version: AI-ADA/88K Version 2.4

Host Computer System: VAXstation 3100 cluster
VMS Version 5.3

Target Computer System: Tadpole TP880V (88100 based VME board)
Bare machine

Customer Agreement Number: 90-03-12-AIT

See Section 3.1 for any additional information about the testing environment.

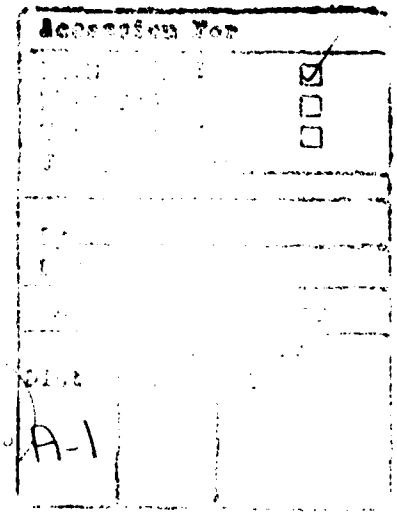
As a result of this validation effort, Validation Certificate 900930W1.11030 is awarded to Aitech Defense Systems Inc. This certificate expires on 1 March 1993.

This report has been reviewed and is approved.

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301



AVF Control Number: AVF-VSR-390.0291
3 February 1991
90-03-12-AIT

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 900930W1.11030
Aitech Defense Systems Inc.
AI-ADA/88K Version 2.4
VAXstation 3100 cluster => Tadpole TP880V (88100 based VME board)

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

aitech

Declaration of Conformance

Customer:	Aitech Defense Systems Inc.
Ada Validation Facility:	ASD/SCEL, Wright-Patterson AFB
ACVC Version:	1.11
Ada Implementation:	
Compiler Name and Version:	AI-ADA/88K Version 2.4
Host Computer System:	VAXstation 3100 cluster, VMS 5.3
Target Computer System:	Tadpole TP880V (88100 based VME board) Bare machine

Customer's Declaration

I, the undersigned, representing Aitech Defense Systems, declare that Aitech Defense Systems has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration.

Date: September 27, 1990

Gabriel Leemor

Aitech Defense Systems Inc.
1250 Oakmead Parkway
Sunnyvale, CA 94086

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementation's is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.

Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 02 September 1990.

E28005C	B28006C	C34006D	B41308B	C43004A	C45114A
C45346A	C45612B	C45651A	C46022A	B49008A	A74006A
B83022B	B83022H	B83025B	B83025D	B83026B	B85001L
C83026A	C83041A	C97116A	C98003B	BA2011A	CB7001A
CB7001B	CB7004A	CC1223A	BC1226A	CC1226B	BC3009B
BD1B02B	BD1B06A	AD1B08A	BD2A02A	CD2A21E	CD2A23E
CD2A32A	CD2A41A	CD2A41E	CD2A87A	CD2B15C	BD3006A
CD4022A	CD4022D	CD4024B	CD4024C	CD4024D	CD4031A
CD4051D	CD5111A	CD7004C	ED7005D	CD7005E	AD7006A
CD7006E	AD7201A	AD7201E	CD7204B	BD8002A	BD8004C
CD9005A	CD9005B	CDA201E	CE2107I	CE2119B	CE2205B
CE2405A	CE3111C	CE3118A	CE3411B	CE3412B	CE3812A
CE3814A	CE3902B				

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

IMPLEMENTATION DEPENDENCIES

C24113I..K (3 tests) contain lines greater than the implementations maximum line length of 126 characters.

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

C45531M..P (4 tests) and C45532M..P (4 tests) use a value for SYSTEM.MAX_MANTISSA of 47 or greater.

C45536A, C46013B, C46031B, C46033B, and C46034B contain 'SMALL representation clauses which are not powers of two or ten.

C45624A..B (2 tests) check that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types; for this implementation, MACHINE_OVERFLOWS is TRUE.

C4A013B contains the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type. This expression would raise an exception. Since the expression must be static, it is rejected at compile time.

B86001Y checks for a predefined fixed-point type other than DURATION.

C96005B checks for values of type DURATION'BASE that are outside the range of DURATION. There are no such values for this implementation.

CA2009C and CA2009F instantiate generic units before their bodies are compiled. This implementation creates a dependence on generic unit as allowed by AI-00408 and AI-00530 such that the compilation of the generic unit bodies makes the instantiating units obsolete.

LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F check for pragma INLINE for procedures and functions.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A840 use representation clauses specifying non-default sizes for access types.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions.

The following 272 tests check for sequential, text, and direct access files:

CE2102A..C (3)	CE2102G..H (2)	CE2102K	CE2102N..Y (12)
CE2103C..D (2)	CE2104A..D (4)	CE2105A..B (2)	CE2106A..B (2)
CE2107A..H (8)	CE2107L	CE2108A..H (8)	CE2109A..C (3)
CE2110A..D (4)	CE2111A..I (9)	CE2115A..B (2)	CE2117A..B (2)
CE2120A..B (2)	CE2201A..C (3)	EE2201D..E (2)	CE2202A
CE2201F..N (9)	CE2203A	CE2204A..D (4)	CE2205A
CE2206A	CE2208B	CE2401A..C (3)	EE2401D
CE2401E..F (2)	EE2401G	CE2401H..L (5)	CE2403A
CE2404A..B (2)	CE2405B	CE2406A	CE2407A..B (2)
CE2408A..B (2)	CE2409A..B (2)	CE2410A..B (2)	CE2411A
CE3102A..C (3)	CE3102F..H (3)	CE3102J..K (2)	CE3103A
CE3104A..C (3)	CE3106A..B (2)	CE3107A..B (2)	CE3108A..B (2)
CE3109A	CE3110A	CE3111A..B (2)	CE3111D..E (2)
CE3112A..D (4)	CE3114A..B (2)	CE3115A	CE3116A
CE3119A	EE3203A	EE3204A	CE3207A
CE3208A	CE3301A	EE3301B	CE3302A
CE3304A	CE3305A	CE3401A	CE3402A
EE3402B	CE3402C..D (2)	CE3403A..C (3)	CE3403E..F (2)
CE3404B..D (3)	CE3405A	EE3405B	CE3405C..D (2)
CE3406A..D (4)	CE3407A..C (3)	CE3408A..C (3)	CE3409A
CE3409C..E (3)	EE3409F	CE3410A	CE3410C..E (3)
EE3410F	CE3411A	CE3411C	CE3412A
EE3412C	CE3413A..C (3)	CE3414A	CE3602A..D (4)
CE3603A	CE3604A..B (2)	CE3605A..E (5)	CE3606A..B (2)
CE3607B..D (3)	CE3704A..F (6)	CE3704M..O (3)	CE3705A..E (5)
CE3706D	CE3706F..G (2)	CE3804A..P (16)	CE3805A..B (2)
CE3806A..B (2)	CE3806D..E (2)	CE3806G..H (2)	CE3904A..B (2)
CE3905A..C (3)	CE3905L	CE3906A..C (3)	CE3906E..F (2)

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 19 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B33301B	B35701A	B38003A	B38003B	B55A01A	B83E01C
B83E01D	B83E01E	BA1001A	BA1101B	BC1109A	BC1109C

IMPLEMENTATION DEPENDENCIES

C85006A..E (5 tests) were each split into 2 tests in which either of approximately half of the lines in the main sequence of statements were commented out. These Test Modifications were necessary because the compiler exhausts the system's memory resources otherwise.

Pragma ELABORATE was added to C83030C and C86007A to force the elaboration of package REPORT.

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For a point of contact for technical information about this Ada implementation system, see:

Gabriel Leemor
1250 Oakmead Parkway
Suite 210
Sunnyvale CA 94086

For a point of contact for sales information about this Ada implementation system, see:

Gabriel Leemor
1250 Oakmead Parkway
Suite 210
Sunnyvale CA 94086

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

PROCESSING INFORMATION

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

Total Number of Applicable Tests	3579
Total Number of Withdrawn Tests	74
Processed Inapplicable Tests	44
Non-Processed I/O Tests	272
Non-Processed Floating-Point Precision Tests	201
Total Number of Inapplicable Tests	517
Total Number of Tests for ACVC 1.11	4170

The above number of I/O tests were not processed because this implementation does not support a file system. The above number of floating-point tests were not processed because they used floating-point precision exceeding that supported by the implementation. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 517 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation and 272 executable tests that use file operations not supported by the implementation. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded onto a disk of a VAX-11/750 computer. The disk was then connected to the cluster, and all files were copied to the VAXstation 3100 disk.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation. Executable modules were downloaded to the target board via a RS-232 serial link using the AI-ADA Cross Debugger.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

Switch	Effect
/lis	Produced a compilation list file.
/progress_report	Prints the compiler passes during compilation.

Test output, compiler and linker listings, and job logs were captured on disk and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"
$MAX_STRING_LITERAL	'"' & (1..V-2 => 'A') & '"'

MACRO PARAMETERS

The following table lists all of the other macro parameters and their respective values.

Macro Parameter	Macro Value
$MAX_IN_LEN	126
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_647
$DEFAULT_MEM_SIZE	2097152
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	MC88000
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	16#0#
$ENTRY_ADDRESS1	16#4#
$ENTRY_ADDRESS2	16#10#
$FIELD_LAST	50
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FILE_TYPE
$FLOAT_NAME	NO_SUCH_TYPE
$FORM_STRING	" "
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	75000.0
$GREATER_THAN_DURATION_BASE_LAST	131073.0
$GREATER_THAN_FLOAT_BASE_LAST	1.80141E+38
$GREATER_THAN_FLOAT_SAFE_LARGE	1.0E308

```
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
                        1.0E308

$HIGH_PRIORITY          23

$ILLEGAL_EXTERNAL_FILE_NAME1
                        \NODIRECTORY\FILENAME

$ILLEGAL_EXTERNAL_FILE_NAME2
                        THIS-FILE-NAME-IS-TOO-LONG-FOR-MY-SYSTEM

$INAPPROPRIATE_LINE_LENGTH
                        -1

$INAPPROPRIATE_PAGE_LENGTH
                        -1

$INCLUDE_PRAGMA1        PRAGMA INCLUDE ("A28006D1.ADA")

$INCLUDE_PRAGMA2        PRAGMA INCLUDE ("B28006F1.ADA")

$INTEGER_FIRST          -32768

$INTEGER_LAST           32767

$INTEGER_LAST_PLUS_1    32768

$INTERFACE_LANGUAGE     ASM

$LESS_THAN_DURATION     -75000.0

$LESS_THAN_DURATION_BASE_FIRST
                        -131073.0

$LINE_TERMINATOR        ASCII.LF

$LOW_PRIORITY           1

$MACHINE_CODE_STATEMENT
                        NULL;

$MACHINE_CODE_TYPE      NO_SUCH_TYPE

$MANTISSA_DOC           31

$MAX_DIGITS             15

$MAX_INT                2147483647

$MAX_INT_PLUS_1         2147483648

$MIN_INT                -2147483648
```

MACRO PARAMETERS

$NAME	NO_SUCH_TYPE_AVAILABLE
$NAME_LIST	MC68020, MC88000, i860
$NAME_SPECIFICATION1	[CRICKETTL.ACVC11.DEVELOPMENT]X2120A.
$NAME_SPECIFICATION2	[CRICKETTL.ACVC11.DEVELOPMENT]X2120B.
$NAME_SPECIFICATION3	[CRICKETTL.ACVC11.DEVELOPMENT]X2120C.
$NEG_BASED_INT	16#F000000E#
$NEW_MEM_SIZE	65535
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	MC68020
$PAGE_TERMINATOR	ASCII.FF
$RECORD_DEFINITION	NEW INTEGER;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	4096
$TICK	0.000001
$VARIABLE_ADDRESS	16#0420#
$VARIABLE_ADDRESS1	16#0424#
$VARIABLE_ADDRESS2	16#0428#
$YOUR_PRAGMA	INTERFACE_PACKAGE

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

1 The Ada Cross Compiler

1.1 The Invocation Command

Command syntax: ADA88K <source-file-spec>

Command parameters: source-file-spec

The text file contains the source text that is to be compiled. If the file type is omitted in the source file specification, the file type ADA is assumed by default. The source-file-spec is required or otherwise the following prompt will appear.

_FILE:

Command qualifiers: /asm_list (default)
/noasm_list

Only when invoked will an assembler file be generated for the compilation unit. The file name corresponds to the name of the compilation unit and suffix. The suffix "_S" is given to a specification and "_B" to a body. The file type is ASM.

/list
/nolist (default)

A source listing will be generated by the compilers. The file name will correspond to the source-file-spec name with the file type .LIS.

If /nolist is active, no source listing is produced,

regardless of any LIST pragmas in the program or any diagnostic messages produced.

/progress
/noprogress (default)

Data about which pass the compiler is performing will be output to SYS$OUTPUT.

/xref
/noxref (default)

Requests that a cross-reference listing be generated. If no severe or fatal errors are found during the compilation, the cross-reference listing is written to list file (see section The Cross-Reference Listing).

/library=<file-spec>

The current sublibrary in which the compilation is performed. If the qualifier is omitted, the sublibrary designated by the logical name ADA_LIBRARY is used as the current sublibrary.

/configuration_file=<file-spec>

Specifies the configuration file to be used by the compiler. If the qualifier is omitted the configuration file designated by the logical name ADA88K_CONFIG is the default.

/trace
/notrace (default)

Specifies whether trace-back information should be generated. For more information, refer to the chapter on Trace-Back Information.

/suppress_all
/nosuppress_all (default)

Requests that generated code should not perform the checking described in the ARM. /SUPPRESS_ALL specifies that no checking at all should be performed at run time. This qualifier does not override pragma SUPPRESS_ALL. The default provides all the checks that the ARM requires.

/save_source (default)
/nosave_source

Specifies whether the source text of the compilation unit is stored in the program library. In case the source text file contains several compilation units, the source text for each compilation unit is stored in the program library. The source texts stored in the program library can be extracted using the PLU (refer to the TYPE command in the Program Library Utility Section).

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to linker documentation and not to this report.

2 The AI-ADA Cross Linker

2.1 The Invocation Command

Command format: ada88k/link <unit-name> [<recompilation-spec>]
<recompilation-spec> ::= <unit-spec>
{+<unit-spec>}
<unit-spec> ::= <unit-name> |
<unit-name>/SPECIFICATION |
<unit-name>/BODY

Command parameters: unit-name

If a link is requested, unit-name must specify a main program which is a library unit found in the current program library, but not necessarily in the current library. The library unit must be a parameterless procedure. If the main procedure has parameters, then the execution of the program is undefined.

If examination of the consequences of recompilations is requested, unit-name specifies a set of program library units for which the consistency will be checked as they appear after the hypothetical recompilations.

The specified unit-name may include wildcard characters, which will be interpreted according to VAX/VMS rules for wildcard characters.
The following applies to the different kinds of unit names, with and without wildcard characters:

a) without wildcard characters.

Unit-name designates the visible unit of the specified name. The designated unit must be a parameterless procedure.

b) with wildcard characters.

Unit-name designates all library units in the current library with names matching the specified unit-name. All types of library units may be designated.

recompilation-spec

A sequence of unit-spec separated by '+' (plus) or ',' (comma). A unit-spec is a unit-name which may have a /SPECIFICATION or /BODY qualifier. If this parameter is given, the linker will analyze the consequences of hypothetical recompilations of the unit or units given by the recompilation specification. The parameter is a list of unit-names (possibly with wildcards) separated by comma or +. Each unit-name may have a positional qualifier indicating whether the body or the specification is to be considered compiled. If a unit-name does not have this qualifier, /SPECIFICATION is assumed.

Command qualifiers: /first_address=<address>
/first_address=20000 (default)

Directs the linker to allocate memory to the application starting at the specified (hexadecimal) address. This qualifier must be used in accordance with the memory allocated to the Run-Time System (refer to Chapter 15, AI-ARTOS Target Configuration Kit, in the AI-ADA/88K User's Guide).

/last_address=<address>
/last_address=3F0000 (default)

This qualifier informs the linker of the (hexadecimal) value of the last address in the target hardware's RAM address space. The memory contained between the END_DATA section and LAST_ADDRESS is used by the AI-ARTOS storage manager to allocate space for stacks and heaps. If a directives file is used, the definition of the Memory_top parameter given in the file will override this qualifier.

/directives_file=<file_spec>

This optional qualifier specifies the name of a "directives" file for the linker. The directives file contains information regarding the memory configuration of the target hardware.

The structure of the directives file is described in Chapter 15, AI-ARTOS Target Configuration Kit, in the AI-ADA/88K User's Guide.

If this qualifier is omitted, the linker will assume that the target has a contiguous RAM address space starting at the (default) value of qualifier FIRST_ADDRESS and ending at LAST_ADDRESS. The order of the sections will be the

default order.

/log[=<file-spec>]
/nolog (default)

Generates a log file in the file name file-spec. If there is no file specification, a log file named unit-name.LOG or LINK.LOG is created in the current default directory. The log file name LINK.LOG is used if the unit-name contains wildcard characters.

If a file specification is given, this file will be used as the log file.

In any case, the name of the log file is displayed on SYS$OUTPUT.

/target=<target-id>

Target_id is an INTEGER value that identifies a particular target and can be checked by the target-dependent parts of the RTS. This qualifier identifies the target hardware on which the executable image will run. It instructs the linker to use the proper configuration module while linking.

As supplied, the Run-Time System is configured to support the Motorola MVME180 and MVME181, and the Tadpole TP880V CPU boards.

/progress
/noprogress (default)

Displays the object file names included in the executable image.

/with_rts
/nowith_rts (default)

This qualifier instructs the linker to create an executable image that includes all the modules of the Run-Time System necessary for the execution of the Ada program. The image created constitutes a complete, self-contained Ada application. The default value of this qualifier, nowith_rts, causes the linker to include only an interface to the RTS in the executable image, on the assumption that the Run-Time System has been previously downloaded, or resides permanently in the target in the form of firmware.

/trace
/notrace (default)

This qualifier is used to enable different tracing functions in the Run-Time System. In the current version, linking with the qualifier /trace enables the exception traceback mechanism which causes the RTS to print a trace for every exception raised (whether it is handled by the program or not). Note that the trace will be meaningful only for the units that were compiled using the qualifier /trace. Otherwise, the code that supports this feature is not generated. Refer to the section on Traceback Information in this guide.

/library=<file_spec>

Specifies the current sublibrary and also the current program library, which consists of the current sublibrary and its ancestor sublibraries.

If the qualifier is omitted, the sublibrary designated by the logical name ADA88K_LIBRARY used as the current library.

/map (default)
/nomap

Generates a map file with a file name corresponding to the executable image name, with the suffix .MAP. Information pertaining to module size and location, and location of variables is detailed in the MAP file.

/external_module=(module1,module2,...)

Instructs the linker to include the object files of the given modules in the link. This qualifier is used when compilation units included in the link contain references to assembly subprograms whose specifications are given using pragma INTERFACE.

The linker searches for the modules in the current default directory, unless a full path name is provided.

/rts_size=<size>
/rts_size=10000 (default)

Determines the maximum size in bytes (hexadecimal) of the Run-Time System. For further information, refer to Chapter 15, AI-ARTOS Target Configuration Kit, in the AI-ADA/88K User's Guide.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

   type INTEGER is range -32768 .. 32767;

   type FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;

   type DURATION is delta 0.000061 range -131072.0 .. 131071.0;

   type SHORT_INTEGER is range -128 .. 127;

   type LONG_INTEGER is range -2147483648 .. 2147483647;

   type SHORT_FLOAT is digits 4 range -3.403E+38 .. -3.403E+38;

   type LONG_FLOAT is digits 15 range
                  -1.79769313486232E+308 .. 1.79769313486232E+308;

end STANDARD;
```

APPENDIX F OF THE Ada STANDARD

This appendix describes the implementation-dependent characteristics of the AI-ADA/88K Cross Compiler System, as required in the Appendix F frame of the Ada Reference Manual (ANSI/MIL-STD-1815A).

F.1 Implementation-Dependent Pragmas

The following implementation dependent pragmas are defined in the compiler:

* suppress_all
* interface_package
* external_subprogram_name

Refer to the section on Implementation-Dependent Pragmas in this guide for further information.

F.2 Implementation-Dependent Attributes

No implementation-dependent attributes are defined for this version.

F.3 Package SYSTEM

The specification of the package SYSTEM:

```
package SYSTEM is
       type ADDRESS is new LONG_INTEGER;
       subtype PRIORITY is INTEGER range 0..23;
                 -- Priority 0  is reserved for the Null_Task
                 -- Priority 24 is reserved for System Tasks
                 -- Priorities 25..31 are for interrupts
       type NAME is (MC68020, MC88000, i860);
       SYSTEM_NAME: constant NAME  := MC88000;
       STORAGE_UNIT: constant := 8;
       MEMORY_SIZE: constant := 2048 * 1024;
       MIN_INT: constant := -2_147_483_647-1;
       MAX_INT: constant := 2_147_483_647;
       MAX_DIGITS: constant := 15;
       MAX_MANTISSA: constant := 31;
       FINE_DELTA: constant:= 2#1.0#E-31;
       TICK: constant := 0.000_001;

       type INTERFACE_LANGUAGE is (ASM, C, RTS);

end SYSTEM;
```

F.4 Representation Clauses

F.4.1 Length Clauses

The following kinds of length clauses are supported:

1. Size specification: T'size

   Supported as described in ARM. For scalar objects residing in the frame, the smallest possible size (in complete bytes) will always be chosen by the compiler.

2. Specification of a collection size: T'storage_size

   Specifies the number of storage units allocated to the collection associated with access type T.

   If a storage_size of 0 is given for an access type, then no collection will be created for objects of the accessed type. This feature can be used to create access types that point to external (non-Ada) data structures.

3. Specification of task size: T'storage_size

   Specifies the number of storage units allocated for each activation of a task of type T. This size includes space for the task's stack, as well as some RTS overhead (approximately 450 bytes).

4. Specification of small for a fixed point type: T'small

   The effect of the length clause is to use this value of small for the representation of values of the fixed point base type.

F.4.2 Enumeration Representation Clause

Enumeration representation clauses may specify representations in the range of the predefined type LONG_INTEGER.

F.4.3 Record Representation Clause

Record representation clauses are supported as detailed in Section 13.4 of the ARM.

F.5 Implementation-Dependent Names for Implementation-Dependent Components

None defined by the compiler.

F.6 Address Clauses

Address clauses are supported for objects (variables or constants) and task entries (linkage to hardware interrupt); refer to Section 15.8, "Connecting External Interrupts to Ada Rendezvous."

Address clauses for objects are interpreted as absolute addresses, and code is generated using the ORG directive. The compiler does not check for possible overlap.

F.7 Unchecked Conversion

No warning is issued when conversion between objects of different sizes is performed. The result of such a conversion is unpredictable.

F.8 Input-Output Packages

Input-Output packages are supplied with the AI-ADA/88K Cross Compiler System.

Standard_input and Standard_output are supported. External files and file objects are implementation-dependent, and therefore are handled as specified in the ARM. F.8.1.Specification of the Package Sequential_IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;
generic
       type ELEMENT_TYPE is private;
package SEQUENTIAL_IO is
       type FILE_TYPE is limited private;
       type FILE_MODE is (IN_FILE, OUT_FILE);

-- File management
procedure CREATE (
       FILE : in out FILE_TYPE;
       MODE : in FILE_MODE  := OUT_FILE;
       NAME : in STRING     := ""; FORM : in
              STRING               := "");

 procedure OPEN (
       FILE : in out FILE_TYPE;
       MODE : in FILE_MODE;
       NAME : in STRING;
       FORM : in STRING      := "");

procedure CLOSE (FILE : in out FILE_TYPE);
procedure DELETE (FILE : in out FILE_TYPE);
procedure RESET (
       FILE : in out FILE_TYPE;
       MODE : in FILE_MODE);

procedure RESET (FILE : in out FILE_TYPE);
```

```
        function MODE   (FILE : in FILE_TYPE) return FILE_MODE;
               function NAME   (FILE : in FILE_TYPE) return STRING;

        function FORM   (FILE : in FILE_TYPE) return STRING;
        function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;
               -- input and output operations

        procedure READ  (
               FILE : in FILE_TYPE;
               ITEM : out ELEMENT_TYPE);
        procedure WRITE (
               FILE : in FILE_TYPE;
               ITEM : in ELEMENT_TYPE);
        function END_OF_FILE ( FILE : in FILE_TYPE)
                      return BOOLEAN;

-- exceptions
        STATUS_ERROR : exception renames
               IO_EXCEPTIONS.STATUS_ERROR;
        MODE_ERROR   : exception renames
               IO_EXCEPTIONS.MODE_ERROR;
        NAME_ERROR   : exception renames
               IO_EXCEPTIONS.NAME_ERROR;
        USE_ERROR    : exception renames
               IO_EXCEPTIONS.USE_ERROR;
        DEVICE_ERROR : exception renames
               IO_EXCEPTIONS.DEVICE_ERROR;
        END_ERROR    : exception renames
               IO_EXCEPTIONS.END_ERROR;
        DATA_ERROR   : exception renames
               IO_EXCEPTIONS.DATA_ERROR;

        private
               type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

        end SEQUENTIAL_IO;
```

F.8.2 Specification for Package Direct Input-Output

```
        with BASIC_IO_TYPES;
        with IO_EXCEPTIONS;
        generic
               type ELEMENT_TYPE is private;
        package DIRECT_IO is

               type FILE_TYPE is limited private;
               type FILE_MODE is (IN_FILE, INOUT_FILE, OUT_FILE);
               type COUNT is range 0..LONG_INTEGER'LAST;
               subtype POSITIVE_COUNT is COUNT range 1..COUNT'LAST;

               -- File management
```

```
procedure CREATE (
       FILE : in out FILE_TYPE;
       MODE : in FILE_MODE  := INOUT_FILE;
       NAME : in STRING     := "";
       FORM : in STRING     := "");
procedure OPEN (
       FILE : in out FILE_TYPE;
       MODE : in FILE_MODE;
       NAME : in STRING;
       FORM : in STRING    := "");

procedure CLOSE (FILE : in out FILE_TYPE);
procedure DELETE (FILE : in out FILE_TYPE);
procedure RESET (FILE : in out FILE_TYPE;
                 MODE : in FILE_MODE);

procedure RESET (FILE : in out FILE_TYPE);
function MODE   (FILE : in FILE_TYPE) return FILE_MODE;
function NAME   (FILE : in FILE_TYPE) return STRING;
function FORM   (FILE : in FILE_TYPE) return STRING;
function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

-- input and output operations

procedure READ  (
       FILE : in FILE_TYPE;
       ITEM : out ELEMENT_TYPE;
       FROM : in POSITIVE_COUNT);

procedure READ  (
       FILE : in FILE_TYPE;
       ITEM : out ELEMENT_TYPE);

procedure WRITE (
       FILE : in FILE_TYPE;
       ITEM : in ELEMENT_TYPE;
       TO   : in POSITIVE_COUNT);

procedure WRITE (
       FILE : in FILE_TYPE;
       ITEM : in ELEMENT_TYPE);

procedure SET_INDEX(
       FILE : in FILE_TYPE;
       TO   : in POSITIVE_COUNT);

function INDEX( FILE : in FILE_TYPE) return
        POSITIVE_COUNT;

function SIZE (FILE : in FILE_TYPE) return COUNT;
function END_OF_FILE(FILE : in FILE_TYPE)
       return BOOLEAN;
```

```
-- exceptions

STATUS_ERROR : exception renames
       IO_EXCEPTIONS.STATUS_ERROR;
MODE_ERROR   : exception renames
       IO_EXCEPTIONS.MODE_ERROR;
NAME_ERROR   : exception renames
       IO_EXCEPTIONS.NAME_ERROR;
USE_ERROR    : exception renames
       IO_EXCEPTIONS.USE_ERROR;
DEVICE_ERROR : exception renames
       IO_EXCEPTIONS.DEVICE_ERROR;
END_ERROR    : exception renames
       IO_EXCEPTIONS.END_ERROR;
DATA_ERROR   : exception renames
       IO_EXCEPTIONS.DATA_ERROR;

private
       type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end DIRECT_IO;
```

F.8.3 Specification of Package Text Input-Output

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;
package TEXT_IO is
type FILE_TYPE is limited private;
type FILE_MODE is (IN_FILE, OUT_FILE);
type COUNT is range 0 .. LONG_INTEGER'LAST;
subtype POSITIVE_COUNT is COUNT range 1 .. COUNT'LAST;

UNBOUNDED: constant COUNT:= 0; -- line and page length
subtype FIELD is INTEGER range 0 .. 50;
subtype NUMBER_BASE is INTEGER range 2 .. 16;
type TYPE_SET is (LOWER_CASE, UPPER_CASE);

-- File Management

procedure CREATE (
       FILE  : in out FILE_TYPE;
       MODE  : in FILE_MODE := OUT_FILE;
       NAME  : in STRING    := "";
       FORM  : in STRING    := "");

procedure OPEN   (
       FILE  : in out FILE_TYPE;
       MODE  : in FILE_MODE;
       NAME  : in STRING;
       FORM  : in STRING    := "");
```

```
        procedure CLOSE  (FILE  : in out FILE_TYPE);
        procedure DELETE (FILE  : in out FILE_TYPE);
        procedure RESET  (
              FILE  : in out FILE_TYPE;
              MODE  : in FILE_MODE);
        procedure RESET  (FILE  : in out FILE_TYPE);
        function  MODE   (FILE  : in FILE_TYPE) return FILE_MODE;
        function  NAME   (FILE  : in FILE_TYPE) return STRING;
        function  FORM   (FILE  : in FILE_TYPE) return STRING;
        function  IS_OPEN(FILE  : in FILE_TYPE) return BOOLEAN;

    -- Control of default input and output files

        procedure  SET_INPUT   (FILE : in FILE_TYPE);
        procedure  SET_OUTPUT  (FILE : in FILE_TYPE);
        function   STANDARD_INPUT   return FILE_TYPE;
        function   STANDARD_OUTPUT  return FILE_TYPE;
        function   CURRENT_INPUT    return FILE_TYPE;
        function   CURRENT_OUTPUT   return FILE_TYPE;

    -- specification of line and page lengths

    procedure SET_LINE_LENGTH  (
          FILE : in FILE_TYPE;
          TO   : in COUNT);

    procedure SET_LINE_LENGTH  (TO   : in COUNT);
    procedure SET_PAGE_LENGTH  (
          FILE : in FILE_TYPE;
          TO   : in COUNT);
    procedure SET_PAGE_LENGTH  (TO   : in COUNT);
    function  LINE_LENGTH      (FILE : in FILE_TYPE)
          return COUNT;
    function  LINE_LENGTH return COUNT;
    function  PAGE_LENGTH (FILE : in FILE_TYPE)
              return COUNT;
    function  PAGE_LENGTH return COUNT;

    -- Column, Line, and Page Control

    procedure NEW_LINE (
          FILE    : in FILE_TYPE;
          SPACING : in POSITIVE_COUNT := 1);

    procedure NEW_LINE (SPACING : in POSITIVE_COUNT := 1);

    procedure SKIP_LINE   (
          FILE    : in FILE_TYPE;
          SPACING : in POSITIVE_COUNT := 1);

    procedure SKIP_LINE   (SPACING : in POSITIVE_COUNT := 1);
    function  END_OF_LINE (FILE : in FILE_TYPE)  return BOOLEAN;
```

```
        function  END_OF_LINE  return BOOLEAN;

        procedure NEW_PAGE    (FILE : in FILE_TYPE);
        procedure NEW_PAGE                          ;

        procedure SKIP_PAGE   (FILE : in FILE_TYPE);
        procedure SKIP_PAGE                         ;
        function  END_OF_PAGE (FILE : in FILE_TYPE)  return BOOLEAN;
        function  END_OF_PAGE  return BOOLEAN;
        function  END_OF_FILE (FILE : in FILE_TYPE)  return BOOLEAN;
        function  END_OF_FILE  return BOOLEAN;

        procedure SET_COL (FILE : in FILE_TYPE;
                     (TO   : in POSITIVE_COUNT);

        procedure SET_COL (TO   : in POSITIVE_COUNT);
        procedure SET_LINE (FILE : in FILE_TYPE;
                 (TO   : in POSITIVE_COUNT);

        procedure SET_LINE  (TO   : in POSITIVE_COUNT);
        function  COL       (FILE : in FILE_TYPE)
               return POSITIVE_COUNT;
        function  COL   return POSITIVE_COUNT;

        function  LINE  (FILE : in FILE_TYPE)
               return POSITIVE_COUNT;
        function  LINE   return POSITIVE_COUNT;

        function  PAGE  (FILE : in FILE_TYPE)
               return POSITIVE_COUNT;
        function  PAGE   return POSITIVE_COUNT;

        -- Character Input-Output

        procedure GET  (
               FILE : in FILE_TYPE;
               ITEM : out CHARACTER);

        procedure GET  (ITEM :  out CHARACTER);

        procedure PUT  (
               FILE : in FILE_TYPE;
               ITEM : in CHARACTER);
        procedure PUT   (ITEM : in CHARACTER);

    -- String Input-Output

        procedure GET  (
               FILE : in FILE_TYPE;
               ITEM : out STRING);

        procedure GET  (ITEM : out STRING);
```

```
        procedure PUT  (
                FILE : in FILE_TYPE;
                ITEM : in STRING);

        procedure PUT  (ITEM : in STRING);
        procedure GET_LINE  (
                FILE : in FILE_TYPE;
                ITEM : out STRING;
                LAST : out NATURAL);

        procedure GET_LINE  (
                ITEM : out STRING;
                LAST : out NATURAL);

        procedure PUT_LINE  (
                FILE : in FILE_TYPE;
                ITEM : in STRING);

        procedure PUT_LINE  (ITEM : in STRING);

        -- Generic Package for Input-Output of Integer Types

        generic

                type NUM is range <>;

        package INTEGER_IO is
                DEFAULT_WIDTH : FIELD        := NUM'WIDTH;
                DEFAULT_BASE  : NUMBER_BASE  :=        10;

                procedure GET  (
                        FILE  : in FILE_TYPE;
                        ITEM  : out NUM;
                        WIDTH : in FIELD := 0);

                procedure GET  (
                        ITEM  :  out NUM;
                        WIDTH : in FIELD := 0);

                procedure PUT  (
                        FILE  : in FILE_TYPE;
                        ITEM  : in NUM;
                        WIDTH : in FIELD := DEFAULT_WIDTH;
                        BASE  : in NUMBER_BASE := DEFAULT_BASE);

                procedure PUT  (
                        ITEM  : in NUM;
                        WIDTH : in FIELD := DEFAULT_WIDTH;
                        BASE  : in NUMBER_BASE := DEFAULT_BASE);

                procedure GET  (
                        FROM  : in STRING;
```

```
        ITEM   : out NUM;
        LAST   : out POSITIVE);

procedure PUT  (
        TO     : out STRING;
        ITEM   : in  NUM;
        BASE   : in  NUMBER_BASE := DEFAULT_BASE);
end INTEGER_IO;

-- Generic Packages for Input-Output of Real Types

generic

type NUM is digits <>;
package FLOAT_IO is

DEFAULT_FORE : FIELD :=              2;
DEFAULT_AFT  : FIELD := NUM'digits - 1;
DEFAULT_EXP  : FIELD :=              3;

procedure GET  (
        FILE   : in FILE_TYPE;
        ITEM   : out NUM;
        WIDTH  : in FIELD := 0);

procedure GET  (
        ITEM   : out NUM;
        WIDTH  : in FIELD := 0);

procedure PUT  (
        FILE : in FILE_TYPE;
        ITEM : in NUM;
        FORE : in FIELD := DEFAULT_FORE;
        AFT  : in FIELD := DEFAULT_AFT;
        EXP  : in FIELD := DEFAULT_EXP);

procedure PUT  (
        ITEM : in NUM;
        FORE : in FIELD := DEFAULT_FORE;
        AFT  : in FIELD := DEFAULT_AFT;
        EXP  : in FIELD := DEFAULT_EXP);

procedure GET  (
        FROM : in STRING;
        ITEM : out NUM;
        LAST : out POSITIVE);

procedure PUT  (TO : out STRING;
        ITEM : in NUM;
        AFT  : in FIELD := DEFAULT_AFT;
        EXP  : in FIELD := DEFAULT_EXP);
end FLOAT_IO;
```

```
generic
type NUM is delta <>;
package FIXED_IO is
DEFAULT_FORE : FIELD := NUM'FORE;
DEFAULT_AFT  : FIELD := NUM'AFT;
DEFAULT_EXP  : FIELD := 0;
procedure GET  (FILE  : in FILE_TYPE;
                ITEM  : out NUM;
                WIDTH : in FIELD := 0);

procedure GET  (
                ITEM  : out NUM;
                WIDTH : in FIELD := 0);

procedure PUT  (
                FILE : in FILE_TYPE;
                ITEM : in NUM;
                FORE : in FIELD := DEFAULT_FORE;
                AFT  : in FIELD := DEFAULT_AFT;
                EXP  : in FIELD := DEFAULT_EXP);

procedure PUT  (ITEM : in NUM;
                FORE : in FIELD := DEFAULT_FORE;
                AFT  : in FIELD := DEFAULT_AFT;
                EXP  : in FIELD := DEFAULT_EXP);

procedure GET (
                FROM : in STRING;
                ITEM : out NUM;
                LAST : out POSITIVE);

procedure PUT  (
                TO   : out STRING;
                ITEM : in NUM;
                AFT  : in FIELD := DEFAULT_AFT;
                EXP  : in FIELD := DEFAULT_EXP);

end FIXED_IO;

--  Generic package for Input-Output of Enumeration Types

generic

type ENUM is (<>);
package ENUMERATION_IO is
DEFAULT_WIDTH   : FIELD    := 0;
DEFAULT_SETTING : TYPE_SET := UPPER_CASE;
procedure GET  (
                FILE : in FILE_TYPE;
                ITEM : out ENUM);
procedure GET  (ITEM : out ENUM);
procedure PUT  (
```

```
                        FILE  : in FILE_TYPE;
                        ITEM  : in ENUM;
                        WIDTH : in FIELD := DEFAULT_WIDTH;
                        SET   : in TYPE_SET   := DEFAULT_SETTING);

    procedure PUT  (
                        ITEM  : in ENUM;
                        WIDTH : in FIELD      := DEFAULT_WIDTH;
                        SET   : in TYPE_SET   := DEFAULT_SETTING);

    procedure GET  (FROM : in STRING;
                        ITEM : out ENUM;
                        LAST : out POSITIVE);

    procedure PUT  (TO : out STRING;
                        ITEM : in ENUM;
                        SET  : in TYPE_SET := DEFAULT_SETTING);

    end ENUMERATION_IO;

-- Exceptions

    STATUS_ERROR : exception renames
           IO_EXCEPTIONS.STATUS_ERROR;
    MODE_ERROR   : exception renames
           IO_EXCEPTIONS.MODE_ERROR;
    NAME_ERROR   : exception renames
           IO_EXCEPTIONS.NAME_ERROR;
    USE_ERROR    : exception renames
           IO_EXCEPTIONS.USE_ERROR;
    DEVICE_ERROR : exception renames
           IO_EXCEPTIONS.DEVICE_ERROR;
    END_ERROR    : exception renames
           IO_EXCEPTIONS.END_ERROR;
    DATA_ERROR   : exception renames
           IO_EXCEPTIONS.DATA_ERROR;
    LAYOUT_ERROR : exception renames
           IO_EXCEPTIONS.LAYOUT_ERROR;

    private

           type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

    end TEXT_IO;
```